SYSTEM SOFTWARE
LAB MANUAL

SEMESTER V

Department of Computer Science and Engineering

College of Engineering, Trivandrum

Vision and Mission of the Institution

Vision

National level excellence and international visibility in every facet of engineering research and education.

Mission

To facilitate quality engineering education to equip and enrich young men and women to meet global challenges in development,innovation and application of technology in the service of humanity.

Vision and Mission of the Department

Vision

To be a centre of excellence in education and research in the frontier areas of Computer Science and Engineering.

Mission

To facilitate quality education and research in Computer Science and Engineering, for transforming students into competent professionals catering to needs of the academia,industry and society.

Program Outcomes

PO1 - Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

PO2 - Problem analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

PO3 -Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

PO4 - Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

PO5 - Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

PO6 -The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

PO7 - Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

PO8 - Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO9 - Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO10 - Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

P011 - Project Management and Finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

P012 - Life-long Learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Course Outcomes

Course Code: CS331
Course Name: System Software Lab

At the end of this course, students will be able to,

CO1 Compare and analyze CPU Scheduling Algorithms like FCFS, Round Robin, SJF, and Priority.

CO2 Implement basic memory management schemes like paging.

CO3 Implement synchronization techniques using semaphores etc.

CO4 Implement banker's algorithm for deadlock avoidance.

CO5 Implement memory management schemes and page replacement schemes and file allocation and organization techniques.

CO6 Implement system software such as loaders, assemblers and macro processor.

CO-PO Mapping

COs	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3
CO1	2							1						2	
CO2	2		2	3				1						2	
CO3	2	2			3			1						2	
CO4	2	3	2					1						2	
CO5	2	2	2		1			1						2	
CO6	3	2	2		3			1						2	

Course Code	Course Name	L-T-P-Credits	Year of Introduction
CS331	System Software Lab	0-0-3-1	2016

Pre-requisite: Nil

Course Objectives

- To build an understanding on design and implementation of different types of system software.

List of Exercises/ Experiments(12 Exercises/ Experiments are to be completed).

Part A

1. Simulate the following non-preemptive CPU scheduling algorithms to find turnaround time and waiting time. a) FCFS b) SJF c) Round Robin (pre-emptive) d) Priority
2. Simulate the following file allocation strategies. a) Sequential b) Indexed c) Linked
3. Implement the different paging techniques of memory management.
4. Simulate the following file organization techniques a) Single level directory b) Two level directory c) Hierarchical
5. Implement the banker's algorithm for deadlock avoidance.
6. Simulate the following disk scheduling algorithms. a) FCFS b)SCAN c) C-SCAN
7. Simulate the following page replacement algorithms a) FIFO b)LRU c) LFU
8. Implement the producer-consumer problem using semaphores.
9. Write a program to simulate the working of the dining philosopher's problem.

Part B

1. Implement the symbol table functions: create, insert, modify, search, and display
2. Implement pass one of a two pass assembler.
3. Implement pass two of a two pass assembler.
4. Implement a single pass assembler.
5. Implement a two pass macro processor.
6. Implement a single pass macro processor.
7. Implement an absolute loader.
8. Implement a relocating loader.
9. Implement pass one of a direct-linking loader.
10. Implement pass two of a direct-linking loader.
11. Implement a simple text editor with features like insertion / deletion of a character, word, and sentence.
12. Implement a symbol table with suitable hashing.

Expected Outcome

The students will be able to

1. Compare and analyze CPU Scheduling Algorithms like FCFS, Round Robin, SJF, and Priority.
2. Implement basic memory management schemes like paging.
3. Implement synchronization techniques using semaphores etc.
4. Implement banker's algorithm for deadlock avoidance.
5. Implement system software such as loaders, assemblers and macro processor.

Evaluation criteria

For Laboratory/ Practical/ Workshop courses:

- **Practical records/ outputs** 60 marks (Internally by the College)
- **Regular class Viva** 10 marks (Internally by the College)
- **Final written test/ quiz** 30 marks (Internally by the College)

All the above assessments are mandatory to earn credits. If not, the student has to complete the course/ assessments during his free time in consultation with the faculty members. On completion of these, grades will be assigned. In case the Practical/ Laboratory/ Workshop courses are not completed in the semester, grade I (incomplete) will be awarded against the course and the final grade will be given only after the completion of the course/ assessments.

Contents

1 Cycle I: Operating System Concepts

1.1 Implementation of Non-preemptive CPU Scheduling Algorithms

1.1.1 Aim

To simulate the following non-preemptive CPU scheduling algorithms to find turnaround time and waiting time. (a) FCFS (b) SJF (c) Round Robin (pre-emptive) (d) Priority.

1.1.2 Algorithm

Algorithm 1: Algorithm for FCFS,SJF,Round Robin,Priority

FCFS Scheduling:

```
Start.
Input the processes along with their burst time (bt) and arrival
  time (at).
Find the waiting time (wt) for all processes.
    As process that arrives first need not wait, wt for that
  process   will be 0, wt[0] = 0.
    Calculate the wt for all other processes as:
    For process i, wt[i] = (bt[0] + bt[1] + .... +bt[i-1]) - at[i]
Find turnaround time (tat) for all processes.
   For process i, tat[i] = wt[i] + bt[i]
Compute average waiting time as (total_wt / no_of_processes).
Compute average turnaround time as (total_tat / no_of_processes).
Stop.
```

SJF Scheduling:

```
Start
Input the processes along with their burst time (bt).
Sort the processes in ascending order of their burst times.
Find the waiting time (wt) for all processes.
       As the process with smallest bt is scheduled first, it need
   not wait,
        so wt for that process will be 0, wt[0] = 0.
       Calculate the wt for all other processes as:
        For process i, wt[i] = wt[i-1] + bt[i-1]
Find turnaround time (tat) for all processes.
    For process i, tat[i] = wt[i] + bt[i]
Compute average waiting time as (total_wt / no_of_processes).
Compute average turnaround time as (total_tat / no_of_processes).
Stop.
```

Algorithm 2: Algorithm for Round Robin,Priority

Round Robin Scheduling:

```
Start
Input the processes along with their burst time (bt).
Input the time quantum, tq.
Create an array rem_bt[] to keep track of the remaining burst time
    of processes. This array is initially a copy of bt[].
Create another array wt[] to store the waiting times of processes.
     Initialize this array as 0.
Initialize time t = 0.
Keep traversing all the processes while all processes are not done
    yet. Do the following for the i th process if it is not done yet
    .
        If rem_bt[i] > tq
           t = t + tq
           rem_bt[i] = tq
        Else
           t = t + rem_bt[i]
           wt[i] = t - bt[i]
           rem_bt[i] = 0
       // Last cycle for this process
      // This process is over
Find turnaround time (tat) for all processes.
   For process i, tat[i] = wt[i] + bt[i]
Compute average waiting time as (total_wt / no_of_processes).
Compute average turnaround time as (total_tat / no_of_processes).
Stop.
```

Priority Scheduling:

```
Start.
Input the processes along with their burst time (bt) and priority.
Sort the processes in ascending order of their priorities.
Find the waiting time (wt) for all processes.
   As the process with highest priority is scheduled first, it need
     not
   wait, so wt for that process will be 0, wt[0] = 0.
  Calculate the wt for all other processes as:
  For process i, wt[i] = wt[i-1] + bt[i-1]
Find turnaround time (tat) for all processes.
  For process i, tat[i] = wt[i] + bt[i]
Compute average waiting time as (total_wt / no_of_processes).
Compute average turnaround time as (total_tat / no_of_processes).
Stop.
```

FCFS:

SAMPLE INPUT:

Enter the number of processes – 3

Enter Burst Time and Arrival Time for Process 0 – 24 0

Enter Burst Time and Arrival Time for Process 1 – 3 0

Enter Burst Time and Arrival Time for Process 2 – 3 0

SAMPLE OUTPUT:

PROCESS BURST ARRIVAL WAITING TURNAROUND

P0	24	0	0	24
P1	3	0	24	27
P2	3	0	27	30

Average Waiting Time – 17.000000

Average Turnaround Time – 27.000000

SJF:

SAMPLE INPUT:

Enter the number of processes – 4

Enter Burst Time for Process 0 – 6

Enter Burst Time for Process 1 – 8

Enter Burst Time for Process 2 – 7

Enter Burst Time for Process 3 – 3

SAMPLE OUTPUT:

PROCESS BURST WAITING TURNAROUND

P3	3	0	3
P0	6	3	9
P2	7	9	16
P1	8	16	24

Average Waiting Time – 7.000000

Average Turnaround Time – 13.000000

ROUND ROBIN:

SAMPLE INPUT:

Enter the no of processes – 3

Enter Burst Time for process 1 – 24

Enter Burst Time for process 2 – 3

Enter Burst Time for process 3 – 3

Enter the time quantum – 3

SAMPLE OUTPUT:

PROCESS BURST WAITING TURNAROUND

P1	24	6	30
P2	3	4	7
P3	3	7	10

Average Turnaround time – 15.666667

Average Waiting time is – 5.666667

PRIORITY:

SAMPLE INPUT:

Enter the number of processes – 5

Enter the Burst Time Priority of Process 0 — 10 3

Enter the Burst Time Priority of Process 1 — 1 1

Enter the Burst Time Priority of Process 2 — 2 4

Enter the Burst Time Priority of Process 3 — 1 5

Enter the Burst Time Priority of Process 4 — 5 2

SAMPLE OUTPUT:

PROCESS PRIORITY BURST WAITING TURNAROUND

P1 1 1 0 1

P4 2 5 1 6

P0 3 10 6 16

P2 4 2 16 18

P3 5 1 18 19

Average Waiting Time is — 8.200000

Average Turnaround Time is — 12.000000

1.2 Implementation of File Organization Techniques

1.2.1 Aim

To simulate the following file organization techniques
(a) Single level directory (b) Two level directory (c) Hierarchical

1.2.2 Algorithm

Single-Level Directory System:

In a single-level directory system, all the files are placed in one directory. There is a root directory which has all files. It has a simple architecture and there are no sub directories. Advantage of single-level directory system is that it is easy to find a file in the directory. A single-level directory has significant limitations, however, when the number of files increases or when the system has more than one user. Since all files are in the same directory, they must have unique names.

Two-Level Directory System:

In the two-level directory system, each user has own user file directory (UFD). The system maintains a master block that has one entry for each user. This master block contains the addresses of the directory of the users. When a user job starts or a user logs in, the system's master file directory (MFD) is searched. When a user refers to a particular file, only his own UFD is searched. This effectively solves the name collision problem and isolates users from one another. Isolation is an advantage when the users are completely independent but is a disadvantage when the users want to cooperate on some task and to access one another's files.

Hierarchical Directory System:

Hierarchical directory structure, also called a tree-structured directory allows users to create their own subdirectories and to organize their files accordingly. A tree is the most common directory structure. The tree has a root directory, and every file in the system has a unique path name. A directory (or subdirectory) contains a set of files or subdirectories.

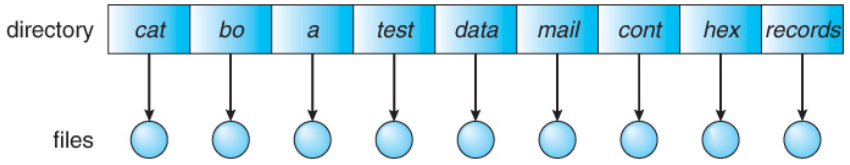


Figure 1: Single-Level Directory System

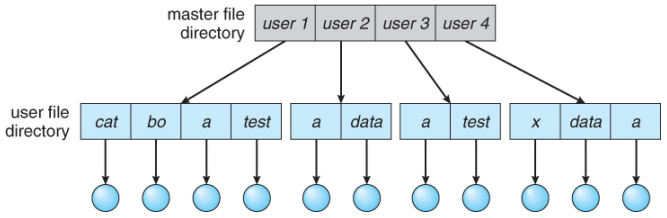


Figure 2: Two-Level Directory System

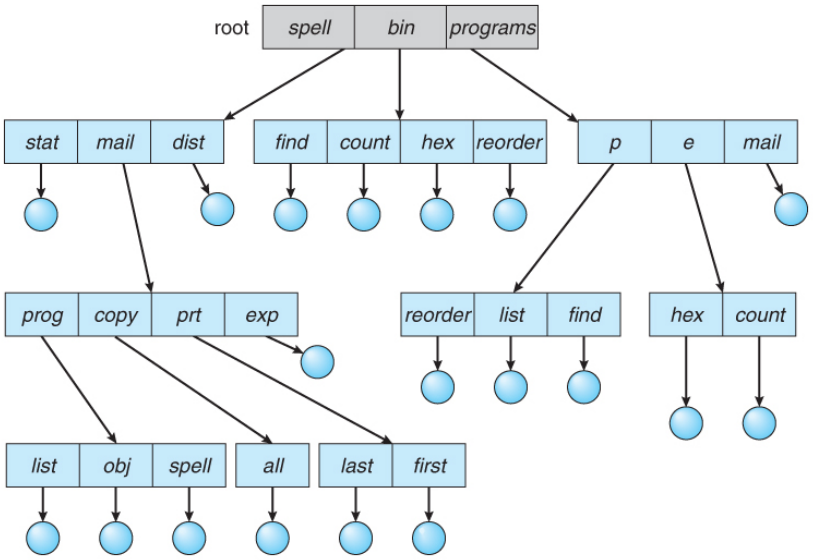


Figure 3: Hierarchical directory system

SAMPLE OUTPUT:

1. Single-level Directory

2. Two-level Directory

3. Hierarchical Directory

4. Exit

Enter your choice: 1

Enter name of directory – CSE

1. Create File 2. Delete File 3. Search File

4. Display Files 5. Exit

Enter your choice – 1

Enter the name of the file – A

1. Create File 2. Delete File 3. Search File

4. Display Files 5. Exit

Enter your choice – 1

Enter the name of the file – B

1. Create File 2. Delete File 3. Search File

4. Display Files 5. Exit

Enter your choice – 1

Enter the name of the file – C

1. Create File 2. Delete File 3. Search File

4. Display Files 5. Exit

Enter your choice – 4

The Files are – A B C

1. Create File 2. Delete File 3. Search File

4. Display Files 5. Exit

Enter your choice – 3

Enter the name of the file – ABC

File ABC not found

1. Create File 2. Delete File 3. Search File

4. Display Files 5. Exit

Enter your choice – 2

Enter the name of the file – B

File B is deleted

1. Create File 2. Delete File 3. Search File

4. Display Files 5. Exit

Enter your choice – 5

Enter your choice: 2

1. Create Directory 2. Create File 3. Delete File

4. Search File 5. Display 6. Exit

Enter your choice – 1

Enter name of directory – DIR1

Directory created

1. Create Directory 2. Create File 3. Delete File

4. Search File 5. Display 6. Exit

Enter your choice – 1

Enter name of directory – DIR2

Directory created

1. Create Directory 2. Create File 3. Delete File

4. Search File 5. Display 6. Exit

Enter your choice – 2

Enter name of the directory – DIR1

Enter name of the file – A1

File created

1. Create Directory 2. Create File 3. Delete File

4. Search File 5. Display 6. Exit

Enter your choice – 2

Enter name of the directory – DIR1

Enter name of the file – A2

File created

1. Create Directory 2. Create File 3. Delete File

4. Search File 5. Display 6. Exit

Enter your choice – 2

Enter name of the directory – DIR2

Enter name of the file – B1

File created

1. Create Directory 2. Create File 3. Delete File

4. Search File 5. Display 6. Exit

Enter your choice – 5

Directory Files

DIR1 A1 A2

DIR2 B1

1. Create Directory 2. Create File 3. Delete File

4. Search File 5. Display 6. Exit

Enter your choice – 4

Enter name of the directory – DIR

Directory not found

1. Create Directory 2. Create File 3. Delete File

4. Search File 5. Display 6. Exit

Enter your choice – 3

Enter name of the directory – DIR1

Enter name of the file – A2

File A2 is deleted

1. Create Directory 2. Create File 3. Delete File

4. Search File 5. Display 6. Exit

Enter your choice – 6

Enter your choice: 3

Enter Name of dir/file (under root): ROOT

Enter 1 for Dir / 2 For File : 1

No of subdirectories / files (for ROOT) :2

Enter Name of dir/file (under ROOT):USER 1

Enter 1 for Dir /2 for file:1

No of subdirectories /files (for USER 1):1

Enter Name of dir/file (under USER 1):SUBDIR

Enter 1 for Dir /2 for file:1

No of subdirectories /files (for SUBDIR):2

Enter Name of dir/file (under USER 1):JAVA

Enter 1 for Dir /2 for file:1

No of subdirectories /files (for JAVA): 0

Enter Name of dir/file (under SUBDIR):VB

Enter 1 for Dir /2 for file:1

No of subdirectories /files (for VB): 0

Enter Name of dir/file (under ROOT):USER2

Enter 1 for Dir /2 for file:1

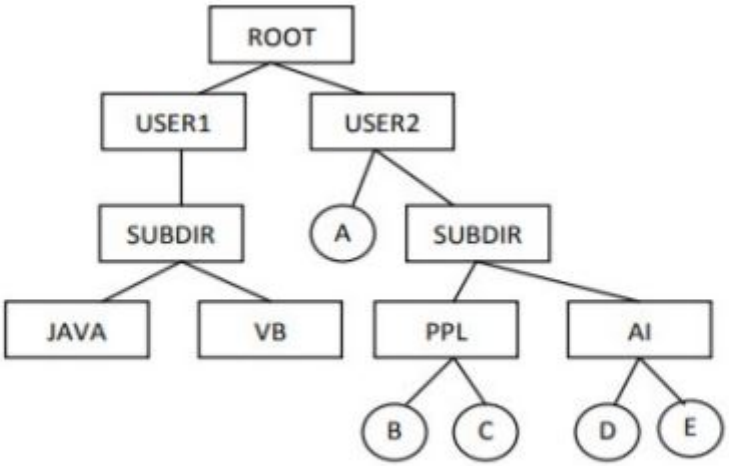
No of subdirectories /files (for USER2):2

Enter Name of dir/file (under ROOT):A

Enter 1 for Dir /2 for file:2

Enter Name of dir/file (under USER2):SUBDIR 2

Enter 1 for Dir /2 for file:1



No of subdirectories /files (for SUBDIR 2):2

Enter Name of dir/file (under SUBDIR2):PPL

Enter 1 for Dir /2 for file:1

No of subdirectories /files (for PPL):2

Enter Name of dir/file (under PPL):B

Enter 1 for Dir /2 for file:2

Enter Name of dir/file (under PPL):C

Enter 1 for Dir /2 for file:2

Enter Name of dir/file (under SUBDIR):AI

Enter 1 for Dir /2 for file:1

No of subdirectories /files (for AI): 2

Enter Name of dir/file (under AI):D

Enter 1 for Dir /2 for file:2

Enter Name of dir/file (under AI):E

Enter 1 for Dir /2 for file:2

1.3 Implementation of Banker's Algorithm

1.3.1 Aim

To implement bankers algorithm, needs the following data structures, where n is the number of processes in the system and m is the number of resource types:

- Available: A vector of length m indicates the number of available resources of each type.
- Max: An n x m matrix defines the maximum demand of each process.
- Allocation: An n x m matrix defines the number of resources of each type currently allocated to each process.
- Need: An n x m matrix indicates the remaining resource need of each process.

1.3.2 Algorithm

Algorithm 3: SAFETY ALGORITHM

```
1  Let Work and Finish be vectors of length m and n, respectively.
       Initialize
2  Work = Available and Finish[i] =false for i = 0, 1, ... , n - 1.
3  Find an index i such that both
4   Finish[i] ==false
5   Needi = Work
6  If no such i exists, go to step 4.
7  Work = Work + Allocation;
8   Finish[i] = true
9   Go to step 2.
10 If Finish[i] ==true for all i, then the system is in a safe state.
```

RESOURCE REQUEST ALGORITHM:

Let Requesti be the request vector for process Pi. If Requesti[j] == k, then process Pi wants k instances of resource type Rj. When a request for resources is made by process Pi, the following actions are taken:

Algorithm 4: RESOURCE REQUEST ALGORITHM

```
If Requesti = Needi, go to step 2. Otherwise, raise an error
    condition,
since the process has exceeded its maximum claim.
If Requesti = Available, go to step 3. Otherwise, Pi must wait,
    since the
resources are not available.
Have the system pretend to have allocated the requested resources
    to
process , Pi by modify the state as follows:
 Available= Available- Requesti ;
 Allocationi = Allocationi + Requesti ;
 Needi = Needi - Requesti ;
```

SAMPLE INPUT:

Enter the number of resources: 4

Enter the maximum instances of each resources:

a=3

b=14

c=12

d=12

Enter the number of processes: 5

Enter the allocation matrix:

a b c d

P[0] 0 0 1 2

P[1] 1 0 0 0

P[2] 1 3 5 4

P[3] 0 6 3 2

P[4] 0 0 1 4

Enter the MAX matrix:

a b c d

P[0] 0 0 1 2

P[1] 1 7 5 0

P[2] 2 3 5 6

P[3] 0 6 5 2

P[4] 0 6 5 6

SAMPLE OUTPUT:

Safe Sequence: $< P[0], P[2], P[3], P[4], P[1] >$

1.4 Implementation of Disk Scheduling Algorithms

1.4.1 Aim

To simulate the following disk scheduling algorithms.
(a) FCFS (b) SCAN (c) C-SCAN

1.4.2 Algorithm

a)FCFS

It is the simplest form of disk scheduling algorithms. The I/O requests are served or processes according to their arrival. The request arrives first will be accessed and served first. Since it follows the order of arrival, it causes the wild swings from the innermost to the outermost tracks of the disk and vice versa. The farther the location of the request being serviced by the read/write head from its current location, the higher the seek time will be.

Algorithm 5: Algorithm for FCFS

```
1 Let Request array represents an array storing indexes of tracks
    that have been requested in ascending order of their time of
    arrival head is the position of disk head.
2 Let us one by one take the tracks in default order and calculate
    the absolute distance of the track from the head.
3 Increment the total seek count with this distance.
4 Currently serviced track position now becomes the new head
    position.
5 Go to step 2 until all tracks in request array have not been
    serviced.
```

SAMPLE INPUT:

Enter the size of Queue 8

Enter the Queue 98, 183,37,122, 14,124,65,67

Enter the initial head position 53

SAMPLE OUTPUT:

Move from 53 to 98 with seek 45

Move from 98 to 183 with seek 85

Move from 183 to 37 with seek 146

Move from 37 to 122 with seek 85

Move from 122 to 14 with seek 108

Move from 14 to 124 with seek 110

Move from 124 to 65 with seek 59

Move from 65 to 67 with seek 2

Total seek time is 640

Average seek time is 80.00000

b)SCAN

This algorithm is performed by moving the R/W head back-and-forth to the innermost and outermost track. As it scans the tracks from end to end, it process all the requests found in the direction it is headed. This will ensure that all track requests, whether in the outermost, middle or innermost location, will be traversed by the access arm thereby finding all the requests. This is also known as the Elevator algorithm.

Algorithm 6: Algorithm for SCAN

```
1 Let Request array represents an array storing indexes of tracks
    that have been requested in ascending order of their time of
    arrival. head is the position of disk head.
2 Let direction represents whether the head is moving towards left
    or right.
3 In the direction in which head is moving service all tracks one by
    one.
4 Calculate the absolute distance of the track from the head.
5 Increment the total seek count with this distance.
6 Currently serviced track position now becomes the new head
    position.
7 Go to step 3 until we reach at one of the ends of the disk.
8 If we reach at the end of the disk reverse the direction and go to
    step 2 until all tracks in request array have not been serviced
    .
```

SAMPLE INPUT:

Enter the size of Queue 8

Enter the Queue 98, 183,37,122, 14,124,65,67

Enter the initial head position 53

SAMPLE OUTPUT:

Move from 53 to 37 with seek 16

Move from 37 to 14 with seek 23

Move from 14 to 0 with seek 14

Move from 0 to 65 with seek 65

Move from 65 to 67 with seek 2

Move from 67 to 98 with seek 31

Move from 98 to 122 with seek 24

Move from 122 to 124 with seek 2

Move from 124 to 183 with seek 59

Total seek time is 236

Average seek time is 29.5

c)C-SCAN

This algorithm is a modified version of the SCAN algorithm. C-SCAN sweeps the disk from end-to-end, but as soon it reaches one of the end tracks it then moves to the other end track without servicing any requesting location. As soon as it reaches the other end track it then starts servicing and grants requests headed to its direction. This algorithm improves the unfair situation of the end tracks against the middle tracks.

Algorithm 7: Algorithm for C-SCAN

```
1 Let Request array represents an array storing indexes of tracks
    that have been requested in ascending order of their time of
    arrival head is the position of disk head.
2 The head services only in the right direction from 0 to size of
    the disk.
3 While moving in the left direction do not service any of the
    tracks.
4 When we reach at the beginning(left end) reverse the direction.
5 While moving in right direction it services all tracks one by one.
6 While moving in right direction calculate the absolute distance of
     the track from the head.
7 Increment the total seek count with this distance.
8 Currently serviced track position now becomes the new head
    position.
9 Go to step 6 until we reach at right end of the disk.
```

SAMPLE INPUT:

Enter the size of Queue 8

Enter the Queue 98, 183,37,122, 14,124,65,67

Enter the initial head position 53

SAMPLE OUTPUT:

Move from 53 to 65 with seek 12

Move from 65 to 67 with seek 2

Move from 67 to 98 with seek 31

Move from 98 to 122 with seek 24

Move from 122 to 124 with seek 2

Move from 124 to 183 with seek 59

Move from 183 to 199 with seek 16

Move from 199 to 0 with seek 199

Move from 0 to 14 with seek 14

Move from 14 to 37 with seek 23

Total seek time is 382

Average seek time is 47.75

1.5 Implementation of Page Replacement Algorithms

1.5.1 Aim

To simulate the following page replacement algorithms a) FIFO b)LRU c) LFU

1.5.2 Algorithm

a) FIFO

It is the simplest page replacement algorithm. In this algorithm, operating system keeps track of all pages in the memory in a queue, oldest page is in the front of the queue. When a page needs to be replaced page in the front of the queue is selected for removal.

Algorithm 8: Algorithm for FIFO

```
Start traversing the pages.
If set holds less pages than capacity.
   Insert page into the set one by one until
     the size  of set reaches capacity or all
     page requests are processed.
   Simultaneously maintain the pages in the
     queue to perform FIFO.
   Increment page fault
Else
  If current page is present in set, do nothing.
  Else
     Remove the first page from the queue
       as it was the first to be entered in
       the memory
     Replace the first page in the queue with
       the current page in the string.
     Store current page in the queue.
Increment page faults.
Return page faults.
```

SAMPLE OUTPUT :

```
Enter page no : 12
Enter values : 2,3,2,1,5,2,4,5,3,2,5,2
Enter page frame: 3
2 -1 -1
2  3 -1
2  3 -1
2  3  1
5  3  1
```

```
5  2  1
5  2  4
5  2  4
3  2  4
3  2  4
3  5  4
3  5  2

 Number of page faults  : 6
```

b) LRU

In Least Recently Used (LRU) algorithm is a Greedy algorithm where the page to be replaced is least recently used. The idea is based on locality of reference, the least recently used page.

Algorithm 9: Algorithm for LRU

```
1  Start traversing the pages.
2   If set holds less pages than capacity.
3     Insert page into the set one by one until
4       the size  of set reaches capacity or all
5       page requests are processed.
6     Simultaneously maintain the recent occurred
7       index of each page in a map called indexes.
8     Increment page fault
9   Else
10   If current page is present in set, do nothing.
11   Else
12     Find the page in the set that was least
13     recently used. We find it using index array.
14     We basically need to replace the page with
15      minimum index.
16      Replace the found page with current page.
17      Increment page faults.
18      Update index of current page.
19 Return page faults.
```

SAMPLE OUTPUT

```
Enter number of frames: 3

Enter number of pages: 6
Enter reference string: 5 7 5 6 7 3

5 -1 -1
5 7 -1
5 7 -1
5 7 6
5 7 6
3 7 6

Total Page Faults = 4
```

c) LFU

Least Frequently Used (LFU) is a type of cache algorithm used to manage memory within a computer. The standard characteristics of this method involve the system keeping track of the number of times a block is referenced in memory.

SAMPLE OUTPUT

```
Enter length of page reference sequence:8

Enter the page reference sequence:2
3 4 2 3 5 6 2

Enter no of frames:3
Enter your choice:5
 For 2 : 2
 For 3 : 2 3
 For 4 : 2 3 4
 For 5 : 2 3 5
 For 6 : 2 3 6
 Total no of page faults:5
```

1.6 Implementation of Producer - Consumer Problem

1.6.1 Aim

To implement the producer-consumer problem using semaphores

1.6.2 Algorithm

Algorithm 10: Algorithm for Producer

```
do
{
 // wait until empty > 0 and then decrement empty
 wait(empty);
 //acquire lock
 wait(mutex);
 /* perform the insert operation in a slot */
 //release lock
 signal(mutex);
 //increment full
 signal(full);
}
while (TRUE)
```

1.6.3 Algorithm

Algorithm 11: Algorithm for Consumer

```
do
{
 // wait until full >0 and then decrement full
 wait(full);
 //acquire the lock
 wait(mutex);
 /* perform the remove operation in a slot */
 //release the lock
 signal(mutex);
 //increment empty
 signal(empty);
}
while (TRUE)
```

SAMPLE OUTPUT:

1.Producer

2.Consumer

3.Exit

Enter your choice:1

Producer produces the item 1

Enter your choice:2

Consumer consumes item 1

Enter your choice:2

Buffer is empty!!

Enter your choice:1

Producer produces the item 1

Enter your choice:1

Producer produces the item 2

Enter your choice:1

Producer produces the item 3

Enter your choice:1

Buffer is full!!

Enter your choice:3

1.7 Implementation of Dining Philosopher's Problem

1.7.1 Aim

To implement a program to simulate the working of the dining philosopher's problem

1.7.2 Algorithm

There is a dining room containing a circular table with five chairs. At each chair is a plate, and between each plate is a single chopstick. In the middle of the table is a bowl of spaghetti. Near the room are five philosophers who spend most of their time thinking, but who occasionally get hungry and need to eat so they can think some more. In order to eat, a philosopher must sit at the table, pick up the two chopsticks to the left and right of a plate, then serve and eat the spaghetti on the plate. Thus, each philosopher is represented by the following pseudo code:

Algorithm 12: Algorithm for Dining Philosopher

```
process P[i]
 while true do
 { THINK;
 PICKUP(CHOPSTICK[i], CHOPSTICK[i+1 mod 5]);
 EAT;
 PUTDOWN(CHOPSTICK[i], CHOPSTICK[i+1 mod 5])
 }
```

SAMPLE OUTPUT:

Philosopher 1 is thinking....

Philosopher 1 is eating....

Philosopher 2 is thinking....

Philosopher 3 is thinking....

Philosopher 3 is eating....

Philosopher 4 is thinking....

Philosopher 5 is thinking....

Philosopher 1 finished eating....

Philosopher 3 finished eating....

Philosopher 5 is eating....

Philosopher 2 is eating....

Philosopher 4 is thinking....

Philosopher 5 finished eating....

Philosopher 2 finished eating....

Philosopher 4 is eating....

Philosopher 4 finished eating....

1.8 Viva Questions:

1. Define operating system.

2. What are the different types of operating systems?

3. Define a process.

4. What is CPU Scheduling?

5. Define arrival time, burst time, waiting time, turnaround time.

6. What is the advantage of round robin CPU scheduling algorithm?

7. Which CPU scheduling algorithm is for real-time operating system?

8. In general, which CPU scheduling algorithm works with highest waiting time?

9. Is it possible to use optimal CPU scheduling algorithm in practice?

10. What is the real difficulty with the SJF CPU scheduling algorithm?

11. What is multi-level queue CPU Scheduling?

12. Differentiate between the general CPU scheduling algorithms like FCFS, SJF etc. and multi-level queue CPU Scheduling.

13. What are CPU-bound I/O-bound processes?

14. What are the parameters to be considered for designing a multilevel feedback queue scheduler?

15. Differentiate multi-level queue and multi-level feedback queue CPU scheduling algorithms.

16. Define file.

17. What are the different kinds of files?

18. What is the purpose of file allocation strategies?

19. Identify ideal scenarios where sequential, indexed and linked file allocation strategies are most appropriate.

20. What are the disadvantages of sequential file allocation strategy?

21. What is an index block?

22. What is the file allocation strategy used in UNIX?

23. What is the purpose of memory management unit?

24. Differentiate between logical address and physical address.

25. What are the different types of address binding techniques?

26. What is the basic idea behind contiguous memory allocation?

27. How is dynamic memory allocation useful in multi programming operating systems?

28. What is fragmentation? Differentiate internal and external fragmentation.

29. Which of the dynamic contiguous memory allocation strategies suffer with external fragmentation?

30. What are the possible solutions for the problem of external fragmentation?

31. What is 50-percent rule?

32. What is compaction?

33. Which of the memory allocation techniques first-fit, best-fit, worst-fit is efficient? Why?

34. What are the advantages of non-contiguous memory allocation schemes?

35. What is the process of mapping a logical address to physical address with respect to the paging memory management technique?

36. Define the terms – base address, offset.

37. Differentiate between paging and segmentation memory allocation techniques.

38. What is the purpose of page table?

39. What is the effect of paging on the overall context-switching time?

40. Define directory.

41. Which of the directory structures is efficient? Why?

42. Which directory structure does not provide user-level isolation and protection?

43. What is the advantage of hierarchical directory structure?

44. What is deadlock? What are the conditions to be satisfied for the deadlock to occur?

45. How can be the resource allocation graph used to identify a deadlock situation?

46. How is Banker's algorithm useful over resource allocation graph technique?

47. Differentiate between deadlock avoidance and deadlock prevention.

48. What is disk scheduling?

49. List the different disk scheduling algorithms.

50. Define the terms – disk seek time, disk access time and rotational latency.

51. What is the advantage of C-SCAN algorithm over SCAN algorithm?

52. Which disk scheduling algorithm has highest rotational latency? Why?

53. Define the concept of virtual memory.

54. What is the purpose of page replacement?

55. Define the general process of page replacement.

56. List out the various page replacement techniques.

57. What is page fault?

58. Which page replacement algorithm suffers with the problem of Belady's anomaly?

59. Define the concept of thrashing. What is the scenario that leads to the situation of thrashing?

60. What are the benefits of optimal page replacement algorithm over other page replacement algorithms?

61. Why can't the optimal page replacement technique be used in practice?

62. What is the need for process synchronization?

63. Define a semaphore.

64. Define producer-consumer problem.

65. Discuss the consequences of considering bounded and unbounded buffers in producer-consumer problem.

66. Can producer and consumer processes access the shared memory concurrently? If not, which technique provides such a benefit?

67. Differentiate between a monitor, semaphore and a binary semaphore.

68. Define clearly the dining-philosophers problem.

2 Cycle II: Assemblers, Loaders and Macro-processors

2.1 Implementation of Pass One of a Two Pass Assembler

2.1.1 Aim

To implement pass one of a two pass assembler.

2.1.2 Algorithm

Assemblers typically make two or more passes through a source program in order to resolve forward references in a program. A forward reference is defined as a type of instruction in the code segment that is referencing the label of an instruction,but the assembler has not yet encountered the definition of that instruction.

Pass 1 assembler reads the entire source program and constructs a symbol table of names and labels used in the program, that is, name of data fields and programs labels and their relative location (offset) within the segment.

Pass 1 determines the amount of code to be generated for each instruction.

Pass-1:

Define symbols and literals and remember them in

symbol table and literal table respectively.

Keep track of location counter

Process pseudo-operations.

Algorithm 13: Algorithm for pass one of a two pass assembler

```
begin
  read first input line
  if OPCODE = 'START' then
    begin
      save #[OPERAND] as starting address
      initialized LOCCTR to starting address
      write line to intermediate file
      read next input line
    end {if START}
  else
    initialized LOCCTR to 0
while OPCODE != 'END' do
    begin
      if this is not a comment line then
        begin
          if there is a symbol in the LABEL field then
            begin
              search SYMTAB for LABEL
              if found then
                set error flag (duplicate symbol)
              else
                insert (LABEL, LOCCTR) into SYMTAB
            end {if symbol}
          search OPTAB for OPCODE
          if found then
            add 3 {instruction lengh} to LOCCTR
          else if OPCODE = 'WORD' then
            add 3 to LOCCTR
          else if OPCODE = 'RESW' then
            add 3 * #[OPERAND] to LOCCTR
          else if OPCODE = 'RESB' then
            add #[OPERAND] to LOCCTR
          else if OPCODE = 'BYTE' then
            begin
              find length of constant in bytes
              add length to LOCCTR
            end {if BYTE}
          else
            set error flag (invalid operation code)
        end {if not a comment}
      write line to intermediate file
      read next input line
    end {while not END}
  write last line to intermediate file
  save (LOCCTR - starting address) as program length
end
```

SAMPLE INPUT:

In SAMPLE.TXT

```
START 1000
FIRST LDA ALPHA
- ADD BETA
- STA GAMA
ALPHA WORD 5
BETA WORD 10
ALPHA WORD 2
BETA RESW 4
GAMA RESW 1
- END
```

SAMPLE OUTPUT:

SYMBOL TABLE

```
label address
FIRST 1000
ALPHA 1009
BETA 1012
GAMA 1030
In OUTPUT.TXT
1000 START 1000
1000 FIRST LDA ALPHA
1003 - ADD BETA
1006 - STA GAMA
1009 ALPHA WORD 5
1012 BETA WORD 10
***duplicate symbol***
ALPHA
***duplicate symbol***
BETA
1030 GAMA RESW 1
- END SYMBOL
Program length is 36.
```

2.2 Implementation of Pass Two of a Two Pass Assembler

2.2.1 Aim

To implement pass two of a two pass assembler

2.2.2 Algorithm

The assembler uses the symbol table that it constructed in Pass 1. Now it knows the length and relative position of each data field and instruction, it can complete the object code for each instruction. It produces .OBJ (Object file), .LST (list file) and cross reference (.CRF) files.

Pass-2:

Generate object code by converting symbolic op-code into respective numeric op-code

Generate data for literals and look for values of symbols

Working of Pass-2:

Pass-2 of assembler generates machine code by converting symbolic machine-opcodes into their respective bit configuration(machine understandable form). It stores all machine-opcodes in MOT table (op-code table) with symbolic code, their length and their bit configuration. It will also process pseudo-ops and will store them in POT table(pseudo-op table).

Algorithm 14: Algorithm for pass two of a two pass assembler

```
begin
  read first input file {from intermediate file}
  if OPCODE = 'START' then
    begin
      write listing line
      read next input line
    end {if START}
  write header record to object program
  initialized first Text record
while OPCODE != 'END' do
    begin
      if this is not a comment line then
        begin
          search OPTAB for OPCODE
          if found then
            begin
              if there is a symbol in OPERAND field then
                begin
                  search SYMTAB for OPERAND
                  if found then
                    store symbol value as operand address
                  else
                    begin
                      store 0 as operand address
                      set error flag (undefined symbol)
                    end
                end {if symbol}
              else
                store 0 as operand address
              assemble the object code instruction
            end {if opcode found}
          else if OPCODE = 'BYTE' or 'WORD' then
            convert constant to object code
          if object code not fit into the current Text record then
            begin
              write Text record to object program
              initialized new Text record
            end
          add object code to Text record
        end {if not comment}
      write listing line
      read next input line
    end {while not END}
```

SAMPLE INPUT:

In INNER.TXT:

```
1000 COPY START 1000
1000 - LDA ALPHA
1003 - ADD BETA
1006 - STA GAMMA
1009 ALPHA WORD 15
100A BETA WORD 15
100C GAMMA RESW 1
- - END -
```

SAMPLE OUTPUT:

SYMTAB.TXT

```
ALPHA 1009
BETA 100A
GAMMA 100C
```

OUTPUT:

```
H^COPY^00^00T^001000^F^031009^16100a^14100c
E^00100
```

2.3 Implementation of Single Pass Assembler

2.3.1 Aim

To implement a single pass assembler

2.3.2 Algorithm:

Algorithm 15: Algorithm for Single Pass Assembler

```
1 Read first line from the intermediate file.
2 Check to see if the opcode from the first line read is START. If so
     then write label,opcode and operand field values of
    corresponding statement directly to final output files.
3 Start the following processing for other lines in intermediate file
     if it is not a comment line until an END statement is reached.
4 Start writing labels LOCCTR opcode and operand fields of
    corresponding statements to the output file along with the
    object code.
5 The object code is found by assembling each statement opcode
    machine equivalent with the label address.
6 If there is no symbol or label in the operand field , then the
    operand address is assigned as zero and it is assembled with
    object code of instruction.
7 If OPCODE is BYTE, WORD, RESB etc are convert constants to object
    code close operand file and exit.
```

SAMPLE INPUT:

```
COPY START 1000
 LDA ALPHA
 STA ALPHA
 ADD BETA
 v STA GAMMA
 EOF BYTE C'EOF'
 . this is a comment
 HEX BYTE X'F0'
 ALPHA WORD 1
 BETA WORD 2
 GAMMA RESW 4096
 END COPY
```

SAMPLE OUTPUT:

```
 H^COPY^001000^003016
 T^001000^10^000000^500000^6B0000^500000^454f46^F0
```

```
T^001001^02^1010
T^001004^02^1010
T^001010^03^000001
T^001007^02^1013
T^001013^03^000002
T^00100a^02^1016
E^001000
```

2.4 Implementation of Two Pass Macro Processor

2.4.1 Aim

To implement a two pass macro processor.

2.4.2 Algorithm:

Algorithm 16: Pass 1 algorithm for two pass macro processor

```
Pass 1 of macro processor makes a line by line scan over its input.
 Set MDTC=1 as well as MNTC=1. Read next line from input program.
If it is a MACRO pseudo-op, the entire macro de_nition except this
(MACRO) line is stored in MDT.
 The name is entered into Macro Name Table along with a pointer to
   the _rst
location of MDT entry of the de_nition.
When the END pseudo-op is encountered all the macro-de_nitions have
    been
processed, so control is transferred to pass2.
```

Algorithm 17: Pass 2 algorithm for two pass macro processor

```
This algorithm reads one line input program at a time.
For each Line it checks if op-code of that line matches any of the
   MNT
entry.
When match is found (i.e. when call is pointer called MDTF to
corresponding macro definition stored in MDT.
The initial value of MDTP is obtained from MDT index field on MNT
entry.
The macro expander prepares the ALA consisting of a table of dummy
argument indices & corresponding arguments to the call.
Reading proceeds from the MDT, as each successive line is read, the
values from the argument list one substituted for dummy arguments
indices in the macro definition.
Reading MEND line in MDT terminates expansion of macro & scanning
continues from the input file.
When END pseudo-op encountered, the expanded source program is
given to the assembler
```

PASS 1:

SAMPLE INPUT:

In minp2.txt (input file)

```
 EX1 MACRO &A,&B
 - LDA &A
 - STA &B
 - MEND -
 SAMPLE START 1000
 - EX1 N1,N2
N1 RESW 1
 N2 RESW 1
 - END -
```

SAMPLE OUTPUT:

In dtab2.txt

```
 EX1 &A,&B
 LDA &A
 STA &B
 MEND
 ntab2.txt
 EX1
```

PASS 2:

SAMPLE INPUT:

minp2.txt

```
 EX1 MACRO &A,&B
 - LDA &A
 - STA &B
 - MEND -
 SAMPLE START 1000
 - EX1 N1,N2
 N1 RESW 1
 N2 RESW 1
 - END -
 dtab2.txt
```

```
EX1 &A,&B
LDA &A
STA &B
MEND
ntab2.txt
EX1
```

SAMPLE OUTPUT:

atab2.txt

```
N1
N2
op2.txt
SAMPLE START 1000
. EX1 N1,N2
- LDA N1
- STA N2
N1 RESW 1
N2 RESW 1
- END -
```

2.5 Implementation of Absolute Loader

2.5.1 Aim

To implement an absolute loader.

2.5.2 Algorithm:

The absolute loader is a kind of loader in which relocated object files are created, loader accepts these files and places them at a specified location in the memory.

This type of loader is called absolute loader because no relocating information is needed, rather it is obtained from the programmer or assembler.

The starting address of every module is known to the programmer, this corresponding starting address is stored in the object file then the task of loader becomes very simple that is to simply place the executable form of the machine instructions at the locations mentioned in the object file.

Algorithm 18: Algorithm for absolute loader

```
begin
read Header record
verify program name and length
read first Text record
while record type != E do
begin
{if object code is in character form, convert into
internal representation}
move object code to specified location in memory
read next object program record
end
jump to address specified in End record
end
```

INPUT:

In text file:

```
H^SAMPLE^001000^0C
T^001000^09^001003^181006^0C1009^
T^002000^03^001010^
E^001000
```

OUTPUT:

```
Enter the program name: SAMPLE

001000 00
001001 10
001002 03
001003 18
001004 10
001005 06
001006 0C
001007 10
001008 09
002000 00
002001 10
002002 10
```

2.6 Implementation of Symbol Table With Hashing

2.6.1 Aim

To implement a symbol table with suitable hashing.

2.6.2 THEORY

Symbol table is an important data structure created and maintained by compilers in order to store information about the occurrence of various entities such as variable names, function names, objects, classes, interfaces, etc.

Symbol table is used by both the analysis and the synthesis parts of a compiler. Each entry in symbol table is associated with attributes that support compiler in different phases.

A symbol table may serve the following purposes depending upon the language in hand:

- To store the names of all entities in a structured form at one place.
- To verify if a variable has been declared.
- To implement type checking, by verifying assignments and expressions in the source code are semantically correct.
- To determine the scope of a name (scope resolution).

Items stored in the symbol table:

- Variable names and constants
- Procedure and function names
- Literal constants and strings
- Compiler generated temporaries
- Labels in source languages

A symbol table is simply a table which can be either linear or a hash table. It maintains an entry for each name in the following format:

$< symbolname, type, attribute >$

The basic operations defined on a symbol table include:

- allocate – to allocate a new empty symbol table free – to remove all entries and free the storage of a symbol table

- insert – to insert a name in a symbol table and return a pointer to its entry
- lookup – to search for a name and return a pointer to its entry
- set attribute – to associate an attribute with a given entry
- get attribute – to get an attribute associated with a given entry

Implementing Symbol Tables with hashing:

Symbol tables are mostly implemented as hash tables, where the source code symbol itself is treated as a key for the hash function and the return value is the information about the symbol. A hash table is an array with index range from 0 to table size - 1.These entries are pointer pointing to names of symbol table. To search for a name we use hash function that will result in any integer between 0 to table size - 1. With hashing, insertion and lookup can be done in O(1) time. Advantage is quick search is possible and disadvantage is that hashing is complicated to implement.

ALGORITHM:

```
Start.
Define the structure of the Symbol Table
Enter the choice for performing the operations in the Symbol Table
If the entered choice is 1, search the symbol table for the symbol to be
    inserted. If the symbol is already present, it displays "Duplicate Symbol",
else, insert the symbol and the corresponding address in the symbol table.
If the entered choice is 2, delete the particular symbol.
If the entered choice is 3, the symbols present in the symbol table are
    displayed.
If the entered choice is 4, the symbol is searched in the symbol table. If it
    is not found in the symbol table it displays "Not found".
If the entered choice is 5, the symbol to be modified is searched in the
    symbol table. The address of the label can be modified.
Enter choice 6 to exit the program.
Stop.
```

OUTPUT:

1.Insert

2.Delete

3.Search

4.Modify

5.Display

6.Exit

Enter the choice: 1

Enter the symbol to be inserted: abc

Enter the datatype: int

Enter the value: 43

1.Insert

2.Delete

3.Search

4.Modify

5.Display

6.Exit

Enter the choice: 5

0

1 [main, int, 2,]

2 [f, int, 2,]

3 [ch, char,1,]

4 [abc, int, 2, 43]

5

6

7 [a, int, 2,][add, float, 4,]

8 [b, int, 2, 340]

9 [c, float, 4,]

1.Insert

2.Delete

3.Search

4.Modify

5.Display

6.Exit

Enter the choice: 2

Enter the symbol to be deleted: add

1.Insert

2.Delete

3.Search

4.Modify

5.Display

6.Exit

Enter the choice: 5

0

1 [main, int, 2,]

2 [f, int, 2,]

3 [ch, char,1,]

4 [abc, int, 2, 43]

5

6

7 [a, int, 2,]

8 [b, int, 2, 340]

9 [c, float, 4,]

1.Insert

2.Delete

3.Search

4.Modify

5.Display

6.Exit

Enter the choice: 2

Enter the symbol to be deleted: s

Symbol Not Found!

1.Insert

2.Delete

3.Search

4.Modify

5.Display

6.Exit

Enter the choice: 3

Enter the symbol to be searched: abc

The symbol is present:

name: abc

value: 43

type: int

size: 2

1.Insert

2.Delete

3.Search

4.Modify

5.Display

6.Exit

Enter the choice: 3

Enter the symbol to be searched: d

Symbol doesn't exist!

1.Insert

2.Delete

3.Search

4.Modify

5.Display

6.Exit

Enter the choice: 4

Enter the symbol to be searched: a

Enter the new value: 10

1.Insert

2.Delete

3.Search

4.Modify

5.Display

6.Exit

Enter the choice: 5

0

1 [main, int, 2,]

2 [f, int, 2,]

3 [ch, char,1,]

4 [abc, int, 2, 43]

5

6

7 [a, int, 2, 10]

8 [b, int, 2, 340]

9 [c, float, 4,]

1.Insert

2.Delete

3.Search

4.Modify

5.Display

6.Exit

Enter the choice: 6

2.7 Implementation of One Pass Macro Processor

2.7.1 Aim

To implement a one pass macro processor

2.7.2 Algorithm

```
begin {macro processor}
 EXPANDING : = FALSE
 while OPCODE ? END do
 begin
 GETLINE
 PROCESSLINE
 end {while}
 end {macro processor}
 procedure PROCESSLINE
 begin
 search NAMTAB for OPCODE
 if found then
 EXPAND
 else if OPCODE = MACRO then
 DEFINE
 else write source line to expanded file
 end {PROCESSLINE}
procedure EXPAND
 begin
 EXPANDING : = TRUE
 get first line of macro definition {prototype} from DEFTAB
 set up arguments from macro invocation in ARGTAB
 write macro invocation to expanded file as a comment
 while not end of macro definition do
 begin
 GETLINE
 PROCESSLINE
 end {while}
 EXPANDING : = FALSE
 end {EXPAND}
procedure GETLINE
 begin
 if EXPANDING then
 begin get next line of macro definition from DEFTAB
 substitute arguments from ARGTAB for positional notation
 end {if}
 else
 read next line from input file
 end {GETLINE}
```

SAMPLE INPUT:
In Input.txt:

```
EX1 MACRO &A,&B
- LDA &A
- STA &B
- MEND -
SAMPLE START 1000
- EX1 N1,N2
N1 RESW 1
N2 RESW 1
- END -
```

SAMPLE OUTPUT:
In Argtab.txt:

```
N1
N2
In Op.txt:
SAMPLE START 1000
. EX1 N1,N2
- LDA ?1
- STA ?2
N1 RESW 1
N2 RESW 1
- END -
```

In Deftab.txt:

```
EX1 &A,&B
LDA ?1
STA ?2
MEND
```

In Namtab.txt:

```
EX1
```

2.8 Viva questions:

1. Define system software.

2. What is an Assembler?

3. What is instruction set?

4. What is direct addressing mode and indirect addressing mode?

5. Differentiate between Assembler and Interpreter.

6. List the types of registers used in a system.

7. What are the instruction formats of SIC/SC?

8. What are Assembler directives or pseudo-instructions?

9. Give some examples for assembler directives.

10. What are functions required in translation of source program to object code.

11. What is forward reference?

12. What are the tree types of records in a simple object program format?

13. What is the information present in a Header record?

14. What is the information present in a Modification record?

15. What is the information present in a Define record?

16.What is the information present in a Refer record?

17. What are functions performed in Pass 1 by a two pass assembler?

18.What are functions performed in Pass 2 by a two pass assembler?

19.Name the data structures used by an assembler.

20. What is OPTAB?

21. What is SYMTAB?

22. What is LOCCTR?

23. Name the addressing modes used for assembling register-to-memory instructions.

24. What is the advantage of register-to-register instructions?

25. What is a re-locatable program?

26. Name the two methods of performing relocation?

27. What are the machine independent assembler features?

28. What does an assembler perform when it encounters LTORG assembler directive?

29. What is LITTAB or What is basic data structure needed to handle literal?

30. Name the symbol defining statements.

31.What is the use of the symbol defining statement EQU?

32. What is the use of the symbol defining statement ORG?

33. What is relative expression?

34. What is absolute expression?

35. List the types of Assemblers.

36. How assemblers handle forward reference instructions?

37. List the types of one pass Assemblers.

38. What is load-and-go assembler?

39. What is multi-pass assembler?

40. What is MASM assembler?

41.What is near jump and far jump?

42. What is a bootstrap loader?

43.What is an absolute loader?

44. What are the disadvantages of an absolute loader or machine dependent loader?

45. What is a bit mask?

46.What is the purpose of the relocation bit in object code of relocation loader or what is a relocation bit?

47. What is external reference?

48. What is EXTDEF?

49.What is EXTREF?

50. What are data structures needed for linking loader?

51.What is the use ESTAB?

52.What is a macro?

53.How does the macro processor help the programmer?

54.What are the two main assembler directives use with macro definitions?

55.What is the logic behind the two-pass macro processor?

56.What are the three main data structures involved in a macro processor?

57.What does the macro definition table contain?

58.What is the purpose of the ARGTAB?

59.How are the ambiguities in parameters avoided in macro processor?

60. What is meant by conditional macro expansion?

61.Define positional parameters.

62. What should be done for recursive macro expansion if the chosen programming language does not support recursion?

63.What is a pre-Processor?

64.What is a line-by-line macro processor?

65. Give any two examples of macro definitions in ANSI C.